Growing and Shrinking Polygons for Random Testing of Computational Geometry Algorithms

Experience Report

Ilya Sergey

ICFP 2016
September 20th, 2016
Polygons
Polygons
Polygons are *Trees*
Polygons are *Trees*

- We can *grow* them;
Polygons are Trees

• We can grow them;
• We can also trim them;
Polygons are *Trees*

- We can *grow* them;
- We can also *trim* them;
- We can use *QuickCheck* to test their properties.
Would you like to run a scenario project this year?
Sure, why not.

Would you like to run a scenario project this year?

Sure, why not.
Scenario Project Requirements
Scenario Project Requirements

- For 2nd year undergrads, team work,
Scenario Project Requirements

- For 2nd year undergrads, team work,
- One week-long,
Scenario Project Requirements

• For 2nd year undergrads, team work,

• One week-long,

• Should involve math and programming,
Scenario Project Requirements

• For 2nd year undergrads, team work,

• One week-long,

• Should involve math and programming,

• Challenging for students, but easy to assess,
Scenario Project Requirements

- For 2nd year undergrads, team work,
- One week-long,
- Should involve math and programming,
- Challenging for students, but easy to assess,
- To be delivered in about a month.
Scenario Project Requirements

• For 2nd year undergrads, team work,

• One week-long,

• Should involve math and programming,

• Challenging for students, but easy to assess,

• To be delivered in about a month.
How many guards do we really need?
How many guards do we really need?

The answer depends on the shape of the gallery.
How many guards do we really need?

The answer depends on the shape of the gallery.
How many guards do we really need?

The answer depends on the shape of the gallery.
How many guards do we really need?
How many guards do we really need?
How many guards do we really need?
How many guards do we really need?
Art Gallery Problem

For a given gallery (polygon), find the *minimal* set of guards’ positions, so together the guards can “see” the *whole* interior.
Art Gallery Problem

For a given gallery (polygon), find the *minimal* set of guards’ positions, so together the guards can “see” the *whole* interior.

NP-hard
Project: Art Gallery Competition

Find the best solutions for a collection of large polygons.
Project: Art Gallery Competition

Find the best solutions for a collection of large polygons.

- 58 vertices
- 5 guards
Organizers’ TODO
Organizers’ TODO

• Problem generator;
  ‣ Polygons with different “features” (convex, rectangular, etc.)
Organizers’ TODO

• Problem generator;
  ‣ Polygons with different “features” (convex, rectangular, etc.)

• Solution checker with online feedback
  ‣ geometric machinery (triangulation, visibility, …)
  ‣ web-server
Organizers’ TODO

• Problem generator;
  ‣ Polygons with different “features” (convex, rectangular, etc.)
• Solution checker with online feedback
  ‣ geometric machinery (triangulation, visibility, …)
  ‣ web-server
• Make sure that it all works.
Organizers’ TODO

- Problem generator;
  - Polygons with different “features” (convex, rectangular, etc.)
- Solution checker with online feedback
  - geometric machinery (triangulation, visibility, …)
  - web-server
- Make sure that it all works.
Organizers’ TODO

- Problem generator;
  - Polygons with different "features" (convex, rectangular, etc.)

- Solution checker with online feedback
  - geometric machinery (triangulation, visibility, …)
  - web-server

- Make sure that it all works.
Organizers’ TODO

• Problem generator;
  ‣ Polygons with different “features” (convex, rectangular, etc.)

• Solution checker with online feedback
  ‣ geometric machinery (triangulation, visibility, …)
  ‣ web-server

• Make sure that it all works.
Growing polygons
Growing polygons

Primitive polygons with specific “features”
Growing polygons

Primitive polygons with specific "features"
Growing polygons

Seed
Growing polygons

Seed
Growing polygons
Growing polygons
Growing polygons
Growing polygons
Growing polygons
Growing polygons
Growing polygons
Growing polygons
Growing polygons
Growing polygons
Growing polygons
Algorithm for growing polygons

1. Pick a primitive polygon;
2. Locate a segment on a host polygon edge;
3. Scale, rotate and attach the primitive; check for self-intersections
4. Repeat
Abstract polygon generator

```scala
trait PolygonGenerator extends GeneratorPrimitives {

  val seeds : List[Polygon]
  val primitives : List[(Int) => Polygon]
  val locate : Double => Option[(Double, Double)]

  val seedFreqs : List[Int]
  val primFreqs : List[Int]

  val generations : Int

  ...
}
```
Abstract polygon generator

```scala
trait PolygonGenerator extends GeneratorPrimitives {
  val seeds : List[Polygon]
  val primitives : List[(Int) => Polygon]
  val locate : Double => Option[(Double, Double)]
  val seedFreqs : List[Int]
  val primFreqs : List[Int]
  val generations : Int

  ...
}
```
Abstract polygon generator

```scala
trait PolygonGenerator extends GeneratorPrimitives {
  val seeds : List[Polygon]
  val primitives : List[(Int) => Polygon]
  val locate : Double => Option[(Double, Double)]

  val seedFreqs : List[Int]
  val primFreqs : List[Int]

  val generations : Int

  ...
}
```
Abstract polygon generator

```scala
trait PolygonGenerator extends GeneratorPrimitives {

  val seeds : List[Polygon]
  val primitives : List[(Int) => Polygon]
  val locate : Double => Option[(Double, Double)]

  val seedFreqs : List[Int]
  val primFreqs : List[Int]

  val generations : Int

  ...
}
```
Generating contest problems

Rectilinear
Generating contest problems

Quasi-convex
Generating contest problems

Crazy
Organizers’ TODO

• Problem generator;
  ‣ Polygons with different “features” (convex, rectangular, etc.)

• Solution checker with online feedback
  ‣ geometric machinery (triangulation, visibility, …)
  ‣ web-server

• Make sure that it all works.
Organizers’ TODO

• Problem generator;
  ‣ Polygons with different “features” (convex, rectangular, etc.)

• Solution checker with online feedback
  ‣ geometric machinery (triangulation, visibility, …)
  ‣ web-server

• Make sure that it all works.
Testing with ScalaCheck
Testing with ScalaCheck

- **Triangulation** of a polygon of size $N$:
  - centre of each triangle lies within a polygon;
  - triangulation generates $N - 2$ (possibly degenerate) triangles;
Testing with ScalaCheck

• **Triangulation** of a polygon of size N:
  • centre of each triangle lies within a polygon;
  • triangulation generates $N - 2$ (possibly degenerate) triangles;

• **Joe-Simpson algorithm for visibility** polygons (VPs):
  • a vertex of a VP is also within the original polygon;
  • every VP’s edge is within the original polygon;
  • a random point within a VP is indeed visible from its origin;
Testing with ScalaCheck

• **Triangulation** of a polygon of size $N$:
  • centre of each triangle lies within a polygon;
  • triangulation generates $N - 2$ (possibly degenerate) triangles;

• **Joe-Simpson algorithm for visibility** polygons (VPs):
  • a vertex of a VP is also within the original polygon;
  • every VP’s edge is within the original polygon;
  • a random point within a VP is indeed visible from its origin;

• **Solution visibility checker**:
  • a refutation (if it exists) is within the original polygon;
  • a refutation for a set of guards is not within any of their VPs;
Testing with ScalaCheck

- **Triangulation** of a polygon of size N:
  - centre of each triangle lies within a polygon;
  - triangulation generates $N - 2$ (possibly degenerate) triangles;

- **Joe-Simpson algorithm for visibility** polygons (VPs):
  - a vertex of a VP is also within the original polygon;
  - every VP’s edge is within the original polygon;
  - a random point within a VP is indeed visible from its origin;

- **Solution visibility checker**:
  - a refutation (if it exists) is within the original polygon;
  - a refutation for a set of guards is not within any of their VPs;

- **Basic algorithm for solving AGP** by Fisk:
  - delivers a solution of size within the boundary $\lfloor N/3 \rfloor$;
  - the solution checker finds no refutations for its result.
How well did that work?
How well did that work?

- **Multiple** bugs (>20) due to inaccurate treatment of floating points;
How well did that work?

• **Multiple** bugs (>20) due to inaccurate treatment of floating points;

• **Several** bugs (≈10) due to misreading of the textbook algorithms or simplifications in their descriptions;
How well did that work?

- **Multiple** bugs (>20) due to inaccurate treatment of floating points;
- **Several** bugs (~10) due to misreading of the textbook algorithms or simplifications in their descriptions;
- **Two** bugs in the Joe-Simpson algorithm itself;
How well did that work?

- **Multiple** bugs (>20) due to inaccurate treatment of floating points;
- **Several** bugs (~10) due to misreading of the textbook algorithms or simplifications in their descriptions;
- **Two** bugs in the *Joe-Simpson* algorithm itself;
- **Bonus:** an undocumented behavior in the *state-of-the-art* CGAL library (written in C++ using exact arithmetics).
How well did that work?

- **Multiple** bugs (>20) due to inaccurate treatment of floating points;
- **Several** bugs (~10) due to misreading of the textbook algorithms or simplifications in their descriptions;
- **Two** bugs in the *Joe-Simpson* algorithm itself;
- **Bonus**: an undocumented behavior in the *state-of-the-art* CGAL library (written in C++ using exact arithmetics).
Testing basic visibility algorithm

- Main component for *checking* arbitrary solutions;
- Original description (1986) has a number of simplifications and is given in pseudocode.
Testing basic visibility algorithm

```scala
import RandomRectilinearPolygonGenerator._

property("All visibility polygons lie within the original polygon") =
  forAll { (p : Polygon) =>
    val guards = p.vertices
    val vps = guards.map(visibilityPolygon(p, _))

    "Every edge of a visibility polygon is within \$\{p\}\" |: {
      val edges = for (vp <- vps; e <- vp.edges) yield e
      edges.forall(p.containsSegment)
    }
  }
```
Testing basic visibility algorithm

```scala
import RandomRectilinearPolygonGenerator._

property("All visibility polygons lie within the original polygon") =

forall { (p : Polygon) =>
  val guards = p.vertices
  val vps = guards.map(visibilityPolygon(p, _))

  "Every edge of a visibility polygon is within \$\{p\}\" | {:

    val edges = for (vp <- vps; e <- vp.edges) yield e
    edges.forall(p.containsSegment)
  }
}
```
Testing basic visibility algorithm

import RandomRectilinearPolygonGenerator._

property("All visibility polygons lie within the original polygon") = 
  forAll { (p : Polygon) =>
    val guards = p.vertices
    val vps = guards.map(visibilityPolygon(p, _))

    "Every edge of a visibility polygon is within \$\{p\}\" |: {
      val edges = for (vp <- vps; e <- vp.edges) yield e
      edges.forall(p.containsSegment)
    }
  }
Testing basic visibility algorithm

```scala
import RandomRectilinearPolygonGenerator._

property("All visibility polygons lie within the original polygon") =
  forAll { (p : Polygon) =>
    val guards = p.vertices
    val vps = guards.map(visibilityPolygon(p, _))

    "Every edge of a visibility polygon is within ${p}" |: {
      val edges = for (vp <- vps; e <- vp.edges) yield e
      edges.forall(p.containsSegment)
    }
  }
```
Bug in Joe-Simpson algorithm

Randomly generated, 260 vertices, guards in every node
Bug in Joe-Simpson algorithm

Randomly generated, 260 vertices, guards in every node
Shrinking Polygons
Trimming Polygons
Reconstructing polygon generation
Reconstructing polygon generation
Reconstructing polygon generation

“Attachment tree”
Trimming polygons
Trimming polygons
Shrinking strategy

• Build the *attachment tree* while constructing a random polygon;
• Construct all its *subtrees*;
• “Render” the corresponding trimmed polygons.
Bug in Joe-Simpson algorithm

Randomly generated, 260 vertices, guards in every node
Bug in Joe-Simpson algorithm

Randomly generated, 260 vertices, guards in every node
Bug in Joe-Simpson algorithm

After trimming, 20 vertices
Bug in Joe-Simpson algorithm

Removed irrelevant guards
Bug in Joe-Simpson algorithm
Bug in Joe-Simpson algorithm
Bug in Joe-Simpson algorithm

...  

`intersectWithWindow(v(i), v(i + 1), s.top, windowEnd) match {`

  `case Some(p) =>`
  `s.push(p)`
  `advance(v, s, i)`

  `case _ =>`
  `scan(v, s, i, windowEnd, ccw)`

`}`

...
Bug in Joe-Simpson algorithm

```
... 
intersectWithWindow(v(i), v(i + 1), s.top, windowEnd) match {
  case Some(p) if !(windowEnd.isDefined && p =~= windowEnd) =>
    s.push(p)
    advance(v, s, i)
  case _ =>
    scan(v, s, i, windowEnd, ccw)
}
... 
```
Implementation effort

- Geometric primitives and procedures: 1,450 LOC
- Server infrastructure: 1,500 LOC
- Testing framework: 350 LOC
Running the Art Gallery Competition
Running the Art Gallery Competition

• 94 participants, 24 teams, 2360 submissions in five days
Running the Art Gallery Competition

• 94 participants, 24 teams, 2360 submissions in five days

• Server running 24/5
  • for different teams, solution processed concurrently
  • no crashes during the week
  • one non-critical bug in server logic (fixed without restarting)
Running the Art Gallery Competition

• **94** participants, **24** teams, **2360** submissions in five days

• Server running 24/5
  • for different teams, solution processed concurrently
  • *no crashes* during the week
  • *one non-critical bug* in server logic (fixed without restarting)

• Most of the solutions: *textbook algorithm* with optimizations;
Running the Art Gallery Competition

- 94 participants, 24 teams, 2360 submissions in five days

- Server running 24/5
  - for different teams, solution processed concurrently
  - no crashes during the week
  - one non-critical bug in server logic (fixed without restarting)

- Most of the solutions: textbook algorithm with optimizations;

- The winning team implemented the state-of-the art algorithm (2014) using linear programming;
To take away
To take away

• FP works great for rapid development of non-trivial and robust geometric applications;
To take away

• FP works great for rapid development of non-trivial and robust geometric applications;

• QuickCheck-ing CG with polygons is feasible and efficient in practice;
To take away

• FP works great for rapid development of non-trivial and robust geometric applications;

• QuickCheck-ing CG with polygons is feasible and efficient in practice;

• Polygons can be grown and trimmed, just like trees.
To take away

• FP works great for rapid development of non-trivial and robust geometric applications;

• QuickCheck-ing CG with polygons is feasible and efficient in practice;

• Polygons can be grown and trimmed, just like trees.

Thanks!